**Write a C program to simulate the following contiguous**

**memory allocation techniques a) Worst-fit**

**b) Best-fit**

**c) First-fit**

#include <stdio.h>

struct Block {

int size;

int allocated;

};

struct File {

int size;

int block\_no;

};

void resetBlocks(struct Block blocks[], int n) {

for (int i = 0; i < n; i++) {

blocks[i].allocated = 0;

}

}

void firstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – First Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

for (int i = 0; i < n\_files; i++) {

files[i].block\_no = -1;

for (int j = 0; j < n\_blocks; j++) {

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

files[i].block\_no = j + 1;

blocks[j].allocated = 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, j + 1, blocks[j].size);

break;

}

}

if (files[i].block\_no == -1) {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

void bestFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – Best Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

for (int i = 0; i < n\_files; i++) {

int bestIdx = -1;

for (int j = 0; j < n\_blocks; j++) {

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

if (bestIdx == -1 || blocks[j].size < blocks[bestIdx].size) {

bestIdx = j;

}

}

}

if (bestIdx != -1) {

blocks[bestIdx].allocated = 1;

files[i].block\_no = bestIdx + 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, bestIdx + 1, blocks[bestIdx].size);

} else {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

void worstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – Worst Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

for (int i = 0; i < n\_files; i++) {

int worstIdx = -1;

for (int j = 0; j < n\_blocks; j++) {

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

if (worstIdx == -1 || blocks[j].size > blocks[worstIdx].size) {

worstIdx = j;

}

}

}

if (worstIdx != -1) {

blocks[worstIdx].allocated = 1;

files[i].block\_no = worstIdx + 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, worstIdx + 1, blocks[worstIdx].size);

} else {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

int main() {

int n\_blocks, n\_files, choice;

printf("Memory Management Scheme\n");

printf("Enter the number of blocks: ");

scanf("%d", &n\_blocks);

printf("Enter the number of files: ");

scanf("%d", &n\_files);

struct Block blocks[n\_blocks];

struct File files[n\_files];

printf("\nEnter the size of the blocks:\n");

for (int i = 0; i < n\_blocks; i++) {

printf("Block %d: ", i + 1);

scanf("%d", &blocks[i].size);

blocks[i].allocated = 0;

}

printf("Enter the size of the files:\n");

for (int i = 0; i < n\_files; i++) {

printf("File %d: ", i + 1);

scanf("%d", &files[i].size);

}

do {

printf("\n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

resetBlocks(blocks, n\_blocks); // Reset block allocation before each strategy

switch (choice) {

case 1:

firstFit(blocks, n\_blocks, files, n\_files);

break;

case 2:

bestFit(blocks, n\_blocks, files, n\_files);

break;

case 3:

worstFit(blocks, n\_blocks, files, n\_files);

break;

case 4:

printf("\nExiting...\n");

break;

default:

printf("Invalid choice.\n");

}

} while (choice != 4);

return 0;

}

Output:

